AJAX Endpoints Are A Big CMS Security BIind Spot

Five AJAX and API vulnerabilities hit Joomla and WordPress in March 2026, all sharing one root cause. Here is what went wrong and how to protect your sites.

Phil E. Taylor | 3 April 2026

In March 2026, five separate AJAX and API vulnerabilities were disclosed across Joomla and WordPress. All five exploited the same weakness: AJAX endpoints that, at most, verify a CSRF token but never check who the user is or whether they have permission to perform the action.

The individual vulnerabilities have been patched. Joomla 5.4.4 added a framework-level authentication check to `com_ajax`, and WordPress's `admin-ajax.php` already requires a login for non-`nopriv` hooks. But neither system checks *authorization* at the framework level. That is still the developer's responsibility, and the pattern of AJAX handlers shipping without capability checks shows that too many developers do not realise it.

What makes AJAX endpoints so dangerous?

Both Joomla's `com_ajax` and WordPress's `admin-ajax.php` were built as lightweight utility endpoints. They exist so plugins and extensions can handle AJAX requests without needing their own dedicated routes. Convenient for developers, dangerous in practice.

The design assumption behind both systems is the same: the plugin on the other end will handle its own security. The AJAX handler just passes the request through. At best, it verifies a CSRF token or nonce to confirm the request was not forged. But a CSRF check is not authentication, and it is definitely not authorization. It proves the request came from a legitimate page. It does not prove who sent it or whether they should be allowed to do what they are asking.

Extension developers treat these endpoints as if they were internal buses, trusting that by the time a request reaches their code, it has already been validated. But `com_ajax` is publicly accessible. `admin-ajax.php` is accessible to any authenticated user, including subscribers. When the plugin on the other end skips its own authorization check, you get an unauthenticated (or under-authenticated) attack surface hiding behind what looks like internal plumbing.

That is not a one-off, and March 2026 proved it five times over.

What is the difference between authentication and authorization?

Most of the vulnerabilities in this post confuse these two concepts, or skip both entirely.

Authentication verifies *who are you?* - is this a real, logged-in user with a valid session? A CSRF token does not do this. It only confirms the request was not forged from another site. Checking authentication means checking the user's session state: `is_user_logged_in()` in WordPress, or `$app->getIdentity()->id` in Joomla (returns `0` for guests).

Authorization verifies *are you allowed to do this?* - even if someone is logged in, they may not have permission for a specific action. A WordPress subscriber should not be able to export `wp-config.php`. A Joomla user without admin privileges should not be able to upload files via a template framework. Checking authorization means an explicit capability check: `current_user_can('manage_options')` in WordPress, or `$user->authorise('core.manage', 'com_templates')` in Joomla.

Most of the AJAX vulnerabilities covered here had a CSRF token check but skipped both authentication and authorization. The request was not forged. Nobody checked who sent it.

Why did Joomla's own team have to harden com_ajax?

The most telling entry in the March 2026 disclosure list is CVE-2026-21629 - and it did not come from a third-party extension. The Joomla Security Strike Team themselves found that `com_ajax`, the component routing AJAX requests for every Joomla plugin, was excluded from the default logged-in-user check in the admin area.

That means `com_ajax` in the backend did not even require a logged-in user by default. The `AdministratorApplication` class explicitly whitelisted `com_ajax` on the same

allowlist as the login page itself, so that pre-login AJAX calls (WebAuthn/passkey flows, CAPTCHA) could work. The consequence: any unauthenticated visitor could reach backend AJAX handlers directly.

Third-party developers building admin AJAX handlers could reasonably assume the framework had already verified the user was authenticated. Every other backend component required a logged-in user. `com_ajax` did not, and nothing in Joomla's developer documentation warned them otherwise. The official docs on `com_ajax` focus entirely on mechanics - URL parameters, method naming, response formats - with no mention that the backend endpoint bypassed the normal login requirement.

That left every extension developer responsible for rolling their own authentication and authorization checks inside every AJAX handler. Some did. Astroid and Novarain did not. And because the gap was invisible - nothing in the framework signalled that the developer needed to add those checks - the same mistake was made independently by multiple development teams.

The severity was rated Low, but the implications are serious. This was not a plugin-level oversight. The framework itself had an authorization gap that every extension built on top of it inherited. The Astroid and Novarain vulnerabilities are the direct, predictable consequences of that gap.

What does the actual Joomla fix look like?

The commit that fixed CVE-2026-21629 is worth reading because it shows exactly how Joomla flipped the default. The fix adds a check at the top of `components/com_ajax/ajax.php`:

```
$unauthorizedAdministratorAccessCheck = (
    $app->isClient('administrator') && $app->getIdentity()->guest
);
```

If the request is hitting the admin-area `com_ajax` and the user is a guest (not logged in), the fix then uses PHP reflection to inspect the target method for a new attribute

called `AllowUnauthorizedAdministratorAccess`. If the method does not carry that attribute, the request is rejected with `JERROR_ALERTNOAUTHOR`.

```
$verifyUnauthorizedAdministratorAccessCheck =
    function ($classOrObject, $method): void {
        $reflection = new ReflectionMethod(
            $classOrObject, $method
        );

        foreach ($reflection->getAttributes() as $attribute) {
            if ($attribute->getName()
                === AllowUnauthorizedAdministratorAccess::class
            ) {
                return;
            }
        }

        throw new RuntimeException(
            Text::_('JERROR_ALERTNOAUTHOR')
        );
    };
```

This check runs for every code path through `com_ajax` - module helpers, plugins, and template helpers all get the same gate. The only way through as a guest is to explicitly opt in with the PHP attribute on your method:

```
#[AllowUnauthorizedAdministratorAccess]
public function onAjaxWebauthn(AjaxEvent $event): void
```

The WebAuthn (passkey) plugin gets this attribute because it legitimately needs to work before login. Everything else is blocked by default.

The default flipped. Before: open to everyone unless the developer adds a check. After: blocked for guests unless the developer explicitly opts in with a PHP attribute. Extensions like Astroid and Novarain would now be blocked automatically, because their AJAX methods do not carry the `AllowUnauthorizedAdministratorAccess` attribute.

Backwards-compatibility break for Joomla extension developers

This fix does not just block handlers that were missing authentication. It blocks any admin AJAX handler that does not authenticate *the Joomla way* - specifically, by having a valid admin session. If a developer was already performing their own authentication checks (API key validation, custom token verification, IP-based restrictions, or any other method that does not go through Joomla's session system), their handler will now be rejected before their code even runs. The framework-level guest check fires first, and if there is no Joomla admin session, the request is denied regardless of what the handler itself would have done.

Developers who were doing the right thing - securing their AJAX handlers properly, just not using Joomla's built-in session system - are now broken alongside developers who were doing nothing at all.

The only escape hatch is the `#[AllowUnauthorizedAdministratorAccess]` attribute, which opts the method back out of the framework check entirely. But adding that attribute also removes the protection for handlers that genuinely were insecure, so developers need to be certain their own checks are solid before opting out.

Extensions with legitimate pre-login needs (like WebAuthn) can opt back in with the attribute. Extensions that were unknowingly exposed (like Astroid and Novarain) get protected automatically. Extensions that had their own security but not via Joomla sessions get caught in the crossfire.

Here is what makes this worse: the official 5.4.4 / 6.0.4 release announcement does not mention this breaking change at all. The security fix is listed as "[20260301] - ACL hardening in com_ajax" with no further detail, no warning to extension developers, and no indication that existing plugins may stop working. The announcement even reassures developers that the upgrade is smooth and most extensions will work with the backwards compatibility plugin enabled.

The actual documentation of this breaking change is buried in the 6.0.4 known issues page on the Joomla developer manual - a page that most extension developers will never check for a point release. It acknowledges the change, notes that performing ACL checks in custom code "was considered a best practice," and points to the WebAuthn plugin as a reference implementation for the new attribute. That is the entirety of the guidance.

A security fix that changes the default behaviour of `com_ajax` for every Joomla extension using admin AJAX handlers, that will silently break any plugin relying on the previous open default, shipped with a one-line entry in the release notes and was documented only in a known-issues sub-page of the migration manual. Any extension developer who updated to 5.4.4 and found their AJAX handlers returning authorization errors would have had no obvious explanation unless they stumbled onto that page.

The security fix is the right call. Shipping a breaking change in a point release with no prominent warning to developers is not.

It took over a decade of Joomla releases (3.0.0 through 5.4.3 and 6.0.0 through 6.0.3) for this default to be corrected. Every extension built during that time inherited the open default, and many shipped without the authentication checks that the framework should have enforced from the start.

The official Joomla docs still teach this insecure pattern

As of April 2026, the Joomla developer manual's own AJAX example component ships with no CSRF token check, no authentication, and no authorization in its AJAX controller. The `AjaxController::divide()` method reads user input directly from the request and returns a response without verifying anything about the caller:

```
public function divide()
{
    $input = $this->app->input;
    $a = $input->get("a", 0, "float");
    $b = $input->get("b", 0, "float");
    // ... no Session::checkToken(), no $user->authorise(), no guest check
    $result = $this->_divide($a, $b);
    echo new JsonResponse($result, "It worked!");
}
```

The JavaScript that calls this endpoint does not send a CSRF token either. Any anonymous visitor can hit `index.php?option=com_ajaxdemo&format=json&task=ajax.divide` and the controller will happily process the request.

It could be argued that this is just a simple frontend public example, and security checks would complicate the teaching. But it is called “AjaxDemo” and it does not even use `com_ajax` - it routes through its own component controller. So the official AJAX example component does not demonstrate AJAX security *or* `com_ajax`.

This particular endpoint only divides two numbers, so the security impact is zero. But this is a *teaching example*. Developers copy these patterns into real extensions that handle file uploads, database writes, and admin operations. If the official documentation demonstrates AJAX without any security checks, it should not be a surprise when developers ship extensions without any security checks.

Would the 5.4.4 fix make this example secure? No. The `com_ajaxdemo` example routes requests through its own component controller (`index.php?option=com_ajaxdemo&task=ajax.divide`), not through `com_ajax`. The 5.4.4 authentication check only applies to requests routed through `com_ajax`. Components with their own controllers are completely unaffected by the fix. A developer who copies this pattern into an admin component will have an unauthenticated endpoint that no framework-level check will catch.

It is not just the example component. The official documentation pages for AJAX plugins, com_ajax, and AJAX in general do not mention authentication or authorization either. Nowhere in Joomla’s AJAX documentation does a developer encounter a warning that their handler needs to verify who is calling it or whether they have permission.

The Astroid and Novarain developers did not invent the insecure pattern. They learned it.

How does WordPress handle this differently?

WordPress takes the opposite default. Its `admin-ajax.php` uses a two-hook system with baked-in authentication:

- `wp_ajax_{action}` only fires if the user is logged in. WordPress checks `is_user_logged_in()` itself, before any plugin code runs.

- `wp_ajax_nopriv_{action}` fires for unauthenticated users. Developers must explicitly register this hook to allow guest access.

In WordPress, a developer has to opt into unauthenticated access. In Joomla before 5.4.4, a developer had to opt into authenticated-only access by coding the check themselves. The defaults were reversed.

Neither system checks authorization (does this user have *permission* to do this specific thing?) - that is still the developer's job on both platforms. But WordPress's framework-level gate catches the most common mistake: forgetting authentication entirely. Joomla's `com_ajax` caught nothing until 5.4.4.

Which five incidents prove the AJAX frontdoor security lax pattern?

All five were disclosed in March 2026.

Astroid Framework for Joomla (CVE-2026-21628, CVSS 10.0)

The Astroid Framework vulnerability is the textbook example. The AJAX endpoint in `Admin.php` used Joomla's `checkToken()` to verify CSRF tokens but never checked whether the requester was actually an administrator. Attackers grabbed the token from the public login page and used it to upload backdoors and install SEO spam plugins. No login required. CVSS 10.0, the maximum possible score.

The fix was a single authorization check: `$user->authorise('core.manage', 'com_templates')`. Standard Joomla ACL that should have been there from the start.

Novarain Framework for Joomla (CVE-2026-21627, CVSS 9.5)

The Novarain/Tassos Framework vulnerability went further. Fully unauthenticated, with no token, no login, and no capability check at all. Joomla's `com_ajax` routed requests to the nrframework plugin, which whitelisted file inclusion as a non-admin task. Attackers could include arbitrary PHP files, delete files, and perform SQL injection

through a single AJAX endpoint. Six weeks after the patch, 46.5% of affected sites in our dataset were still running vulnerable versions.

Smart Slider 3 for WordPress (CVE-2026-3098, CVSS 6.5)

The Smart Slider 3 vulnerability targeted the WordPress side. The plugin's export AJAX actions had a nonce for CSRF protection but no capability check. Any subscriber-level user could call the export function and download arbitrary files from the server, including `wp-config.php` with database credentials and authentication keys. Smart Slider 3 has over 800,000 active installs, and roughly 500,000 were still running vulnerable versions at disclosure.

Same pattern, different CMS. CSRF protection without authentication or authorization.

Joomla core com_ajax ACL hardening (CVE-2026-21629)

Covered in detail above. The framework routing every Joomla plugin's AJAX requests was itself missing the default authentication check in the admin area.

Joomla webservice endpoint access bypass (CVE-2026-23899)

Joomla's webservice API layer did not properly verify that incoming requests had the right permissions, allowing unauthorized access to restricted endpoints. Reported by vnth4nhnt from CyStack and fixed in the same release. Different mechanism than `com_ajax`, same category of failure: an endpoint that accepts requests it should reject.

Is this just a March 2026 problem?

No. The same AJAX authorization failure has been producing critical vulnerabilities for at least a year:

Advanced Custom Fields: Extended **(CVE-2025-13486, CVSS 9.8)** - The `acfe/form/render_form_ajax` endpoint had no nonce check and no capability check. Unauthenticated visitors could invoke arbitrary PHP functions via

`call_user_func_array()`, leading to full remote code execution. Over 100,000 WordPress sites affected. Actively exploited in the wild from December 2025.

King Addons for Elementor **(CVE-2025-8489, CVSS 9.8)** - The `king_addons_user_register` AJAX action accepted a `user_role` parameter from POST data without restriction and without any capability check. Any unauthenticated visitor could create an administrator account. Wordfence recorded over 48,400 blocked exploitation attempts from October 2025.

Gravity Forms **(CVE-2025-12352, CVSS 9.8)** - The `copy_post_image` AJAX function performed no file type validation and no authorization check. Unauthenticated attackers could upload PHP backdoors. This is one of the most widely-used premium WordPress form plugins.

Anti-Malware Security and Brute-Force Firewall **(CVE-2025-11705, CVSS 6.5)** - A security plugin with the same vulnerability it was supposed to protect against. The scan AJAX handler had CSRF protection via a nonce but never called `current_user_can()` to verify the user's role. Any subscriber could obtain the nonce through the quarantine view and use it to read arbitrary files, including `wp-config.php`. Over 100,000 installs.

HUSKY Products Filter for WooCommerce **(CVE-2025-1661, CVSS 9.8)** - The `woof_text_search` AJAX action was registered for unauthenticated users via `wp_ajax_nopriv_`. The `template` parameter accepted user input for file inclusion with no path restriction. Unauthenticated local file inclusion leading to full remote code execution.

Same blueprint every time. The March 2026 cluster just made the pattern impossible to ignore by hitting both CMS platforms in the same month.

What should extension developers be doing better?

The five March 2026 CVEs and the year of AJAX vulnerabilities before them all trace back to one habit: developers treating AJAX endpoints as internal plumbing rather than public attack surface.

Always authorize, never just authenticate

A CSRF token or nonce only proves the request was not forged. It says nothing about who the user is or whether they have permission to perform the action. Every AJAX handler needs both an authentication check (is this a valid, logged-in user?) and a capability check (does this user have the right permissions?).

In WordPress, that means `current_user_can('manage_options')` (or whatever capability fits the action) before executing any logic. In Joomla, that means `$user->authorise('core.manage', 'com_yourcomponent')` or a more specific ACL check.

Never trust the framework to do your security for you

Until Joomla 5.4.4, `com_ajax` in the backend did not require authentication. Even after the fix, neither Joomla nor WordPress checks authorization at the framework level. Your AJAX handler is a public endpoint. Treat it like one.

Register the minimum access level

In WordPress, only register `wp_ajax_nopriv_{action}` if the action genuinely needs to work for unauthenticated visitors. For admin-only actions, only register `wp_ajax_{action}`. This gives you the framework's login check as a free first line of defence.

In Joomla (post-5.4.4), the backend `com_ajax` now requires authentication by default. But developers supporting older Joomla versions should still verify the user is logged in (e.g. checking `$app->getIdentity()->id` is non-zero) as a fallback.

Validate and sanitize all input

File paths, role names, SQL fragments - nothing from the request body should be used directly. The King Addons privilege escalation happened because the plugin accepted `user_role=administrator` from POST data without question. The HUSKY vulnerability happened because a file path was used in an inclusion without sanitization.

Audit your existing AJAX handlers

If you maintain a WordPress plugin or Joomla extension, search your codebase for `wp_ajax_`, `wp_ajax_nopriv_`, and `com_ajax` handlers. For each one, verify: does it check the user's capabilities? Does it validate all input? Could a subscriber (or a guest) reach it?

The Astroid fix was a single line of code. The Novarain fix was similar. These are not hard problems to solve - they are easy problems that nobody solved until attackers found them first.

What else was fixed in Joomla 5.4.4?

The Joomla 5.4.4 and 6.0.4 security release on 31 March 2026 patched six CVEs in total. Beyond the AJAX and API issues already discussed:

CVE	Component	Severity	Description	Reporter
CVE-2026-23898	com_joomlaupdate	High	Arbitrary file deletion via the autoupdate server mechanism	Phil Taylor
CVE-2026-21630	com_content webservice	High impact	SQL injection in the articles API via improperly built order clauses	GitHub Security Lab, CyStack
CVE-2026-21631	com_associations	Moderate	XSS in the multilingual associations comparison view	UNC Pembroke researchers
CVE-2026-21632	Various article outputs	Moderate	XSS from unescaped article titles in multiple locations	Peter Vanderhulst

CVE-2026-23898 - Reported by Phil Taylor

CVE-2026-23898 is a high-severity arbitrary file deletion vulnerability in `com_joomlaupdate`, reported by Phil Taylor (the author of this blog and founder of mySites.guru). The autoupdate

server mechanism lacked input validation, allowing an attacker to delete any file on the server that the PHP process has access to - not just Joomla files, but anything the web server user can reach. The component responsible for keeping Joomla up to date had a security hole in it.

An attacker who can delete files can take a site offline, weaken its security configuration, or expose protected directories by removing `.htaccess` or `web.config`. File deletion vulnerabilities can also be chained with other attacks to escalate to full site takeover.

CVE-2026-21630 is another high-impact endpoint vulnerability - SQL injection in the content API. That makes four of the six CVEs in this release targeting API or AJAX-adjacent attack surfaces.

How does mySites.guru help when vulnerabilities like these drop?

Every hour between a security release and your update is an hour your sites are exposed with a known, published vulnerability.

The mySites.guru dashboard shows every connected site's core version and extension versions at a glance. Filter by version to see exactly which sites need updating, without logging into each one individually.

Need to know which sites run a specific vulnerable extension? The extension search indexes every plugin and extension across your entire portfolio and groups results by version number. This is how we flagged 724 agencies running vulnerable Smart Slider 3 and 8,297 sites with the Novarain Framework within hours of disclosure.

Once you know what needs patching, the mass update tool pushes updates to all affected sites in a single operation. For agencies managing dozens or hundreds of sites, this turns a full day's work into a ten-minute task.

After updating, the security audit verifies each site by checking for modified core files, backdoors, and anything else that looks wrong. If a site was compromised before you could apply the patch, the audit catches the signs.

Not sure whether your sites are at risk? Run a free audit on any Joomla or WordPress site to get an immediate assessment.

What should agency owners do right now?

1. Update everything

If you run Joomla sites, update to 5.4.4 or 6.0.4 today. If you run WordPress sites, make sure Smart Slider 3 is on 3.5.1.34 or later. Check for the Astroid Framework (update to 3.3.13+) and the Novarain Framework (update to 6.0.38+) on your Joomla sites.

If you manage multiple sites, mySites.guru can push updates across your entire portfolio from a single dashboard.

> **Joomla 5.4.4 may break existing plugins**
>
> The com_ajax ACL fix in 5.4.4 blocks all unauthenticated admin AJAX requests by default. If you have extensions that use admin AJAX endpoints without a Joomla session (including extensions that had their own authentication via API keys or custom tokens), those extensions will stop working after the update. Extensions that never had any authentication will also stop working, which is the intended outcome. Check for broken functionality in your admin AJAX workflows after updating. Extension developers need to either add the `#[AllowUnauthorizedAdministratorAccess]` attribute to methods that legitimately need unauthenticated access, or ensure their handlers work within a valid Joomla admin session.

2. Audit your AJAX endpoints

Check what your extensions expose through **`com_ajax`** on Joomla or **`admin-ajax.php`** on WordPress. Look at which webservice endpoints are accessible and whether they are properly locked down. The mySites.guru security audit catches many of these issues, but manual review of your extension stack is also worthwhile for custom or niche extensions.

3. Remove what you don't use

Every installed extension is potential attack surface. If you have plugins or extensions you are not actively using, uninstall them. An extension you forgot about can still expose AJAX endpoints to the public internet.

4. Monitor for compromise

Set up real-time file change monitoring so you are alerted immediately if any watched files are modified. Run the suspect content scanner to check for backdoors that may have been installed before you patched.

5. Keep watching

This pattern is not going away. Joomla 5.4.4 added authentication to `com_ajax` and WordPress already gates `wp_ajax_` hooks behind a login check, but neither framework verifies authorization. That is still the developer's job. Every new extension that ships an AJAX handler without a capability check is a potential vulnerability, and the only fix is developer education.

Further reading

- Joomla 6.0.4 and 5.4.4 Security and Bugfix Release - the official release announcement
- Joomla Security Centre - full details on all six CVEs
- CVE-2026-21629: com_ajax ACL hardening - the framework-level fix that confirms the pattern
- Joomla 6.0.4 known issues (B/C break) - the buried documentation of the com_ajax breaking change
- com_ajax fix commit - the actual code change that flipped the default
- Joomla AjaxDemo example - the official AJAX example with no security checks
- Joomla AJAX plugin example - plugin AJAX example, also no auth/authz guidance
- Joomla com_ajax docs - com_ajax reference, no security warnings

- Joomla AJAX general docs - general AJAX docs, no auth/authz mention
- OWASP AJAX Security Cheat Sheet - general AJAX security guidance
- Astroid Framework Vulnerability - CVE-2026-21628 (CVSS 10.0), the textbook com_ajax auth bypass
- Novarain Framework Vulnerability - CVE-2026-21627 (CVSS 9.5), fully unauthenticated com_ajax exploitation
- Smart Slider 3 Vulnerability - CVE-2026-3098 (CVSS 6.5), the WordPress admin-ajax.php equivalent
- Agency Security Guide - broader CMS security guidance for agencies

Frequently Asked Questions

Why are AJAX endpoints a security problem in WordPress and Joomla?

Both WordPress (admin-ajax.php) and Joomla (com_ajax) were designed as lightweight pass-throughs for plugins and extensions to handle AJAX requests. They authenticate the request but rarely authorize the action. Extensions rely on them without implementing their own permission checks, creating unauthenticated or under-authenticated attack surfaces hidden behind what looks like internal plumbing.

What AJAX vulnerabilities were disclosed in March 2026?

Five separate vulnerabilities exploiting AJAX or API pass-throughs were disclosed in March 2026: Astroid Framework (CVE-2026-21628, CVSS 10.0) and Novarain Framework (CVE-2026-21627, CVSS 9.5) via Joomla's com_ajax, Smart Slider 3 (CVE-2026-3098, CVSS 6.5) via WordPress admin-ajax.php, Joomla core com_ajax ACL hardening (CVE-2026-21629), and a Joomla webservice endpoint access bypass (CVE-2026-23899).

What is the difference between authentication and authorization in AJAX handlers?

A nonce or CSRF token is designed to prevent cross-site request forgery, not to verify identity or permissions. It confirms the request originated from a legitimate page, not that the user behind it is authorized to perform the action. Most AJAX vulnerabilities in March 2026 had a valid CSRF check but no authentication or authorization check behind it. The request was not forged, but nobody verified who sent it or whether they had permission.

What was wrong with Joomla's com_ajax component itself?

Joomla's own Security Strike Team found that com_ajax was excluded from the default logged-in-user check in the admin area (CVE-2026-21629). This meant the framework routing AJAX requests for every Joomla plugin had the same authorization gap as the vulnerable plugins built on top of it. The problem ran all the way down to the foundation, not just individual extensions.

Does mySites.guru detect sites affected by these AJAX vulnerabilities?

Yes. mySites.guru indexes every extension on every connected site and cross-references versions against vulnerability databases twice daily. It flagged Astroid Framework, Novarain Framework, and Smart Slider 3 automatically. For Joomla core issues, the dashboard shows every site's version so you can filter by 5.4.3 or 6.0.3 and push the update across your entire portfolio.

How can agencies protect their sites from AJAX endpoint vulnerabilities?

Update all CMS cores, plugins, and extensions immediately when patches drop. Use a tool like mySites.guru to track versions across your portfolio. Audit which extensions expose AJAX endpoints and whether they implement their own authorization. Disable unused extensions. For Joomla, pay particular attention to com_ajax handlers. For WordPress, review admin-ajax.php actions for proper capability checks.

Are WordPress sites affected by the same AJAX security pattern as Joomla?

Yes. WordPress's admin-ajax.php has the same architectural problem as Joomla's com_ajax. Smart Slider 3 (CVE-2026-3098) had CSRF protection on its export AJAX action but no authentication or capability check. Any subscriber could download wp-config.php. Advanced Custom Fields Extended (CVE-2025-13486, CVSS 9.8) and King Addons for Elementor (CVE-2025-8489, CVSS 9.8) had identical root causes. Both CMS platforms share this design pattern.

Why did Joomla extension developers keep making this mistake?

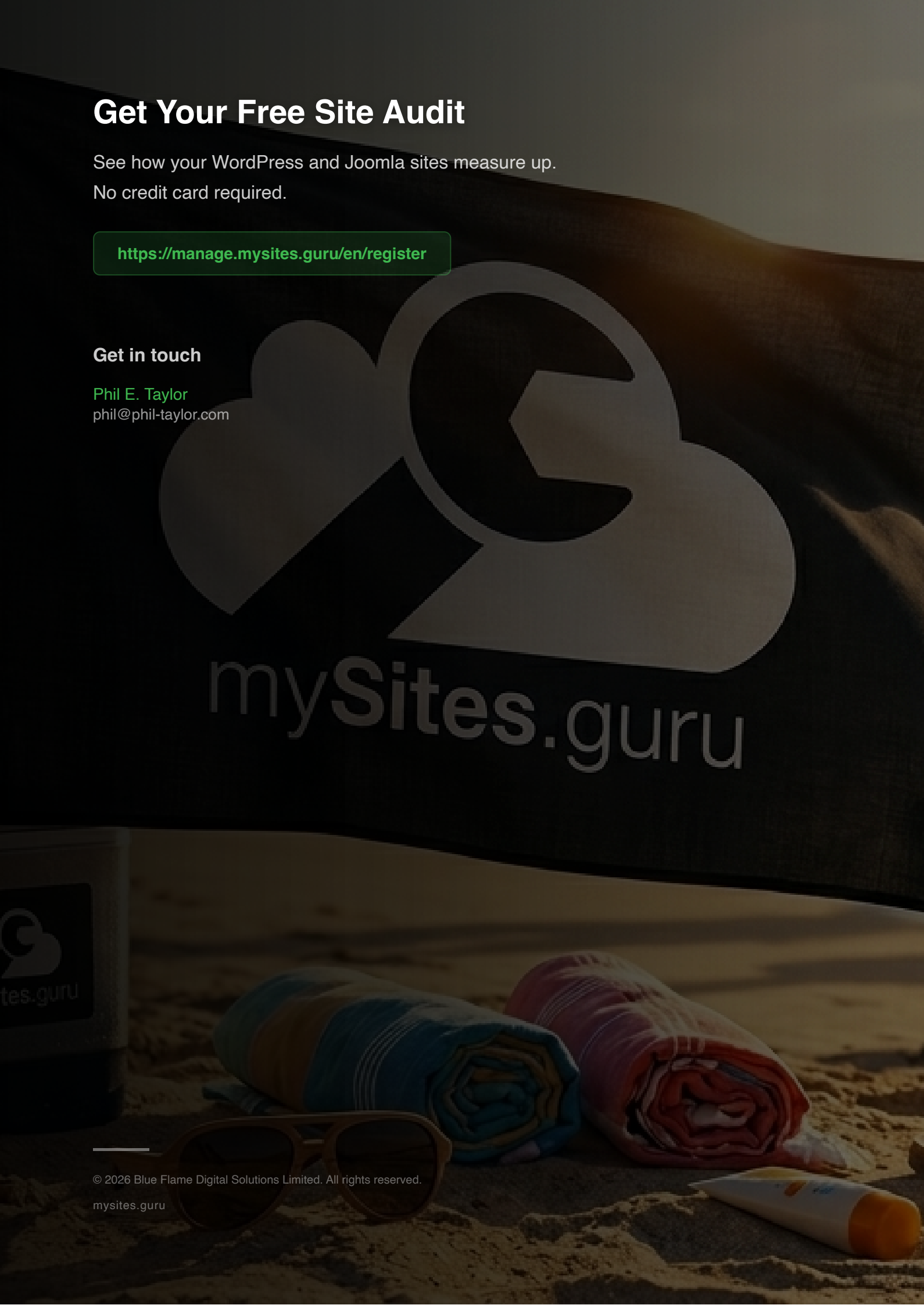
Until Joomla 5.4.4, the com_ajax component in the backend was excluded from the default logged-in-user check. Extension developers building admin AJAX handlers could reasonably assume the framework had already verified the user was authenticated, because every other backend component required a login. Nothing in Joomla's developer documentation warned otherwise. Developers had to roll their own authentication and authorization for every AJAX handler, and many did not realise it.

Get Your Free Site Audit

See how your WordPress and Joomla sites measure up.

No credit card required.

https://manage.mysites.guru/en/register



Get in touch

Phil E. Taylor
phil@phil-taylor.com